Efficient Implementation of

a Low Cost Object Tracking System

by

Asha Sasikumar

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2015 by the
Graduate Supervisory Committee:

Chaitali Chakrabarti, Chair
Umit Ogras
Antonia Papandreou-Suppappola

ARIZONA STATE UNIVERSITY

December 2015

ABSTRACT

Object tracking is an important topic in multimedia, particularly in applications such as teleconferencing, surveillance and human-computer interface. Its goal is to determine the position of objects in images continuously and reliably. The key steps involved in object tracking are foreground detection to detect moving objects, clustering to enable representation of an object by its centroid, and tracking the centroids to determine the motion parameters.

In this thesis, a low cost object tracking system is implemented on a hardware accelerator that is a warp based processor for SIMD/Vector style computations. First, the different foreground detection techniques are explored to figure out the best technique that involves the least number of computations without compromising on the performance. It is found that the Gaussian Mixture Model proposed by Zivkovic gives the best performance with respect to both accuracy and number of computations. Pixel level parallelization is applied to this algorithm and it is mapped onto the hardware accelerator.

Next, the different clustering algorithms are studied and it is found that while DBSCAN is highly accurate and robust to outliers, it is very computationally intensive. In contrast, K-means is computationally simple, but it requires that the number of means to be specified beforehand. So, a new clustering algorithm is proposed that uses a combination of both DBSCAN and K-means algorithm along with a diagnostic algorithm on K-means to estimate the right number of centroids. The proposed hybrid algorithm is shown to be faster than the DBSCAN algorithm by ~2.5x with minimal loss in accuracy. Also, the 1D Kalman filter is implemented assuming constant acceleration model. Since the computations involved in Kalman filter is just a set of recursive equations, the

sequential model in itself exhibits good performance, thereby alleviating the need for parallelization. The tracking performance of the low cost implementation is evaluated against the sequential version. It is found that the proposed hybrid algorithm performs very close to the reference algorithm based on the DBSCAN algorithm.

To my parents, for they inspired me to trust my heart, believe in myself and follow my dreams.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Dr. Chaitali Chakrabarti for her continuous guidance and support. I have been amazingly fortunate to have an advisor who gave me the freedom to explore on my own and at the same time the guidance to recover when my steps faltered. I cherish all my discussions and interactions with her, and am deeply indebted to her for her patience and tolerance towards my imperfections. I am also grateful to my committee members, Dr. Antonia and Dr. Umit for their time in reviewing my work. Also, I would like to thank Dr. Trevor Mudge, University of Michigan, Ann Arbor for his support. I acknowledge DARPA for funding this work.

I am thankful to Qi Zheng and Yajing Chen at University of Michigan, for providing me timely support as and when I needed without any hitch. I am also quite grateful to my friends at Low Power Systems lab, Hsing-min Chen, Ming Yang, Manqing Mao, Jiang Xiang and Siyuan Wei for their help in finishing my thesis.

I am profoundly indebted to my friend, Shrikant for his unconditional support throughout. I am also thankful to my friends Abhishek Badki, Benzun and Kuldeep for their time and patience to clarify my queries. I would like to thank my roommates Karthika, Subha and Manasa for putting up with me during my stay at ASU.

I take this opportunity to thank my parents and my sister, who have been my pillars of support and have encouraged me in all my endeavors. Lastly, I am extremely thankful to Srijith, without whose help I couldn’t have completed this thesis and who was an invaluable source of support at a time when I needed it the most.

TABLE OF CONTENTS

Page

LIST OF TABLES v

LIST OF FIGURES vi

CHAPTER

1 INTRODUCTION 1

Problem Description 2

Proposed Approach 3

Thesis Organization 5

2. BACKGROUND 6

Introduction 6

Object Tracking Methods Based on Matching Followed by Tracking 6

Object Tracking Methods not Based on Matching followed by Tracking 11

Comparison of the Different Techniques for Object Tracking 13

3 PROPOSED OBJECT TRACKING FRAMEWORK 15

Motivation 15

Overview 15

Detailed Description 16

4 RESULTS 32

Simulation Framework 2

Description of the Hardware Accelerator 33

Parallelization Schemes 34

Tracking Performance 38

CHAPTER Page

Timing Results.....42

5 CONCLUSION AND FUTURE WORK 48

REFERENCES.....50

LIST OF TABLES

Table Page

1. Best Configuration for each Video 42

2. Performance of DBSCAN on Intel Core I7 45

3. Performance of DBSCAN on Hardware Accelerator 45

4. Performance of Kalman Filter on Hardware Accelerator.....46

5. Overall Performance of Object Tracking System on Intel Core I7.....47

LIST OF FIGURES

Figure Page

1. Proposed Object Tracking System....16
2. DBSCAN Algorithm....22
3. Proposed Clustering Algorithm....27
4. The Discrete Kalman Filter Cycle....29
5. Loop Unrolling by a Factor of 2....35
6. Variation in Speedups for Gaussian Filter with Loop Unrolling Factor....36
7. Speedup for Foreground Detection using Different Parallelization Schemes....37
8. Tracking Performance for Multi_Car Video; Frame Parameter 10....39
9. Tracking Performance for Multi_Car Video; Frame Parameter 12....40
10. Tracking Performance for Highway Video; Frame Parameter 10....40
11. Tracking Performance for Highway Video; Frame parameter 12....41
12. Tracking Performance for Aerial_highway Video; Frame parameter 10....41
13. Tracking Performance for Aerial_highway Video; Frame parameter 12....42
14. Input Video Frame to find the Best Foreground Detection Technique....43
15. Variation in Execution Times of Foreground Detection on Intel core I7....44
16. Result of Foreground Detection Algorithm....44
17. Runtimes of the Different Kernels on Intel I7....46

CHAPTER 1

INTRODUCTION

Object tracking is an interesting and a challenging topic in the field of computer vision. Its goal is to determine the position of the object in the images continuously and reliably. It is used in the areas of automated surveillance, traffic monitoring, vehicle navigation and motion-based recognition [3]. For instance, in object tracking in video, a tracker assigns consistent labels to the tracked objects in different frames of a video. Additionally, depending on the tracking domain, a tracker can also provide object-centric information, such as orientation, area, or shape of an object.

Tracking objects can be a challenging task in scenes that are prone to illumination changes, object to scene occlusions and cluttering [1]. It can also be challenging when there is loss of information caused by projection of the 3D world on a 2D image [3]. Added to all these algorithm level challenges are the challenges due to real-time processing requirements.

Object tracking can be simplified by imposing constraints on the motion and/or appearance of objects. For example, almost all tracking algorithms assume that the object motion is smooth with no abrupt changes. One can further constrain the object motion to be of constant velocity or constant acceleration based on a priori information. Prior knowledge about the number and the size of objects, or the object appearance and shape, can also be used to simplify the problem.

Numerous approaches for object tracking have been proposed [3]. These primarily differ from each other based on the way they approach the following questions: Which object representation is suitable for tracking? Which image features should be used? How

should the motion, appearance, and shape of the object be modeled? The answers to these questions depend on the context/environment in which the tracking is performed and the end use for which the tracking information is being sought.

1.1 PROBLEM DESCRIPTION

The goal of our research is to design a low cost object tracking system optimized for parallel implementation. Specifically, we address the problem of automated object tracking of fast-moving objects in a video sequence captured from a stationary camera with the additional objective of parallelizing the algorithms in order that they can be efficiently implemented on a hardware accelerator. The task of object tracking is subdivided into (i) foreground detection that involves detecting the moving objects in the frame, (ii) clustering to group the detected objects so that they can be represented by centroids, and (iii) tracking the centroids to estimate the parameters of motion of the object.

In an earlier work by University of Michigan researchers, object tracking was implemented on a warp based Single Instruction Multiple Threads (SIMT) coprocessor. This is a GPGPU like accelerator that has 8 virtual warps; each warp consists of 32 threads that operate on 32 lanes of data simultaneously. The lanes each have their own integer register file, a separate floating point register file, and conditional flags.

The specific object tracking algorithm assumed that the background was not stationary, so additional steps were implemented to remove the effect of the moving background. In our work, we implement a low cost object tracking algorithm on the SIMT based hardware accelerator that assumes that the background is stationary.

1.2 PROPOSED APPROACH

In this work, we explore different object tracking techniques. First, we investigate the performance of different techniques for foreground detection, clustering and tracking by implementing them on an Intel Core I7 processor. We then pick the algorithm that gives the least execution time without compromising on the accuracy and implement the same on the hardware accelerator. We also propose a new clustering algorithm that uses a combination of both DBSCAN [6] and K-means algorithm [35] along with the diagnostic algorithm on K-means to estimate the right number of centroids.

The input to our system is in form of video sequences that are captured from stationary cameras. As a result, the background is considered stationary and the only noise that can occur is due to changes in illumination and any other background clutter such as moving of tree leaves, rain drops, etc. The first step in our procedure, foreground detection involves identifying the moving objects in a frame. The different techniques for this step include frame differencing, mean/variance over time, statistical methods using one or more Gaussians and non-parametric methods [2]. Of these techniques, background subtraction, Gaussian filter and several variants of Mixture of Gaussians were analyzed for performance and execution times. We saw that the mixture of Gaussians developed by Zivkovic [4] gave the best performance in terms of robustness and execution time.

The second step in our procedure is to determine the number of moving objects in a given frame. In order to do this, we use clustering algorithms on the output binary images from the foreground detection step. Though supervised techniques outperform unsupervised clustering techniques, we decided to go for unsupervised techniques since

they are computationally less complex. The various unsupervised clustering techniques that were analyzed include K-means clustering [35], connected components labeling [27], Density Based Spatial Clustering of Applications with Noise (DBSCAN) [6]. Of these, connected components and DBSCAN don't need the number of centroids to be specified beforehand, whereas K-means requires it to be specified. All the clustering algorithms are highly sequential in nature, giving very little scope for parallelization.

We propose an algorithm for clustering that is significantly faster than the conventional clustering algorithms. Basically, we chose a combination of K-means and DBSCAN algorithms with additional diagnostic algorithms to reduce computational complexity. DBSCAN was chosen over connected components because it has inherent noise rejection capabilities. The worst case complexity of DBSCAN is $O(n^2)$ whereas that of K-means is O(kn) where n is the number of data points. In our approach, DBSCAN is performed for one frame and then K-means is performed over the next 9 frames. The number of centroids which is input to K-means is based on the number of centroids returned by DBSCAN. A diagnostic algorithm is developed based on the notion of distance between centroids of adjacent frames that returns the correct number of centroids from K-means. The assumptions are that the object velocity cannot be more than certain orders of magnitude from the frame velocity and also that the number of objects entering/ leaving the frame at a time cannot be more than a certain number. The proposed clustering algorithm gave a speedup of 4.9 compared to DBSCAN.

The final step of our procedure is tracking. Tracking involves estimating the parameters of motion of the object, such as the velocity and the prediction for location of centroid in the next frame. For tracking, we considered the simplest 1-D Kalman filter

that is the least computationally intensive. Each moving object is assigned a kalman filter and this assignment tends to remain constant over frames. The algorithm is in itself very simple and since the sequential algorithm had very minimal execution time, there was no need for any parallelization.

In our attempt to parallelize the proposed object tracking algorithm, we tried parallelization techniques such as those based at the pixel level and those that exploit spatial locality of cache lines by operating on a group of pixels at a time. For instance, for a Gaussian filter of size 5x5, since adjacent pixels along the row of an image are stored in sequential memory locations, loop unrolling along the columns gave better performance than along the rows. For GMM, we found that the smaller loop unrolling factor gave better timing performance and pixel level parallelization gave the best timing performance. The DBSCAN could not be parallelized since the hardware accelerator could not support the memory requirements. The sequential execution time of the Kalman filter was so less that there was no need to parallelize it.

1.3 THESIS ORGANIZATION

The thesis is organized as follows: Chapter 2 gives a detailed description of the present day state-of-the-art techniques for object tracking and also discusses their advantages and disadvantages. Chapter 3 describes the proposed object tracking technique in detail; and Chapter 4 discusses the results that we obtained. We provide our conclusions and the scope for future work in Chapter 5.

CHAPTER 2

BACKGROUND

2.1 INTRODUCTION

Object tracking can be classified into feature, model and optical flow based approaches. Feature based approach involves extraction of regions of interest (features) in the images and then identification of counterparts of individual images of the sequence. Feature based tracking methods include Multi Hypothesis tracking [29], Hidden Markov Model [30], Artificial Neural Network [31], Kalman Filtering [13] and Mean shift [32]. Model based approach is very similar to feature based tracking; the difference is in the requirement of grouping, reasoning and rendering. Additionally, prior knowledge of investigated models is normally required. Optical flow based methods are used for generating dense flow fields by computing the flow vector of each pixel under the brightness constancy constraint. This computation is carried out either algebraically or geometrically.

2.2 OBJECT TRACKING METHODS BASED ON MATCHING FOLLOWED BY TRACKING

2.2.1 Object Tracking with SIFT Features and Mean Shift [1]

This method uses a scale invariant feature transform (SIFT) [33] based mean shift algorithm for object tracking in real scenarios. SIFT features are used to establish correspondence between the regions of interests across frames. Mean shift is applied to conduct similarity search via color histograms. Maximum likelihood estimation of similar regions is achieved by evaluating the probability distributions from these two measurements in an expectation-maximization scheme.

The main steps in this algorithm are: a) Similarity measure by mean shift, b) SIFT feature based correspondence, and c) integration of SIFT and Mean-shift based similarity measure.

a) Similarity measure by Mean shift:

The measurement task involves the search of a confidence region for the target candidate that is most similar to the target model, given the predicted target's position. The similarity measure conducted here is based on color information. Given the sample points and kernel function $k(x)$, the kernel density function can be used to estimate the probability density function of the object in the current image. Similarly, the target image's probability density function can also be estimated. Then correspondence between the feature and its counterparts can be established using feature-spatial space. Either Kullback-Leibler divergence or the Bhattacharya distance can be used to measure the affinity between two distributions.

Mean shift is an instance of gradient ascent with an adaptive step size. Each iteration of mean shift is guaranteed to get closer to a stationary point; however it can get stuck at a saddle point or incorrectly assume a start point at a local minimum for a local maximum. The problem faced by this approach is that there can be a number of discontinuities which can be avoided by taking infinitesimal steps for moving the direction of the local gradient. But if step size is too large, the rate of convergence cannot be guaranteed. The employment of SIFT feature correspondence is a possible optimal solution to this problem.

b) SIFT Feature based Correspondence

SIFT stands for Scale Invariant Feature Transform. SIFT features are distinctive invariant features from images that can be used to perform reliable matching between different views of an object or scene. The features are invariant to image scale and rotation, and are shown to provide robust matching across a substantial range of affine distortion, change in 3D viewpoint, addition of noise, and change in illumination.

c) SIFT and mean shift-based similarity measure

In this stage, an expectation maximization algorithm is applied. An expectation (E) step consists of evaluating the posterior probabilities for each mixture component. A maximization (M) step then updates the mixture components.

The entire algorithmic flow can be summarized as:

(1) Define a rectangle on the region of interest in the first frame of a video sequence.

(2) Compute the color histogram of this region, whilst extracting SIFT features within this region

(3) In the second frame, start from the former location and examine the surroundings for similarity measure. The sum of squared difference (SSD) method is applied for SIFT feature correspondence across frames.

(4) Launch the proposed Expectation Maximization (EM) algorithm to search for an appropriate similarity region whilst minimizing the distance between the detected locations by mean shift and SIFT correspondence, respectively.

(5) Iterate the above steps till the difference between two mean shifts is smaller than a threshold (i.e., 0.01).

In summary, this technique involves establishing correspondence between regions of interests across frames using SIFT feature, applying mean shift to conduct similarity

search based on color histograms and Expectation Maximization to achieve maximum likelihood estimation of similar regions. Using SIFT and mean shift clustering makes this technique quite robust. However, it also makes it computationally intensive.

2.2.2. Consensus-Based Matching and Tracking of Keypoints [28]

This is a keypoint method for long-term model-free object tracking in a combined matching-and tracking framework. A voting mechanism is used wherein each keypoint casts a vote for the object centre. A consensus based scheme is used for outlier detection.

Given a sequence of images $I_1, \ldots, I_n$, and an initializing region b_1 in I_1, the aim in each frame of the sequence is to recover the position of the object of interest or to indicate that the object is not visible. The position of the object is estimated up to its center μ, its scale s and the degree of its in-plane rotation α, where s and α are estimated with respect to the initial appearance of the object.

Matching and Tracking: The model is based on a set of keypoints. Each keypoint denotes a location *r* and descriptor *f*. Binary descriptors are employed to simplify computations. The object model O is initialized by detecting and describing keypoints in I_1 that are inside the initializing region b_1, followed by a mean-normalization of the keypoint locations. Matching and tracking keypoints are two complementary strategies for finding the keypoints. The candidate keypoints in I_t that are determined by their absolute position *a* and their descriptor *f* are detected and described. For each candidate keypoint, its Hamming distance with another keypoint is calculated by XORing the respective descriptors. If P is the set of candidate keypoints, the keypoints in P are matched to keypoints in I_1 by requiring that the nearest neighbor must be closer than the second-nearest neighbor by a certain ratio ρ. The set of matched keypoints M then

consists of the subset of keypoint locations in P that match to O, augmented with the corresponding model keypoint index. Candidate keypoints that match to background keypoints are excluded from M. For tracking, the displacement of each keypoint in K_{t-1} from I_{t-1} to I_t is computed by employing the pyramidal variant of the method of Lucas and Kanade for estimating optical flow. For t = 2, K_1 is obtained by transforming O to absolute image coordinates. The set of tracked keypoints T is then obtained by updating the keypoint locations in K_{t-1} while maintaining the keypoint index.

Voting: In this step, each keypoint in K casts a single vote for the object centre, resulting in a set of votes V. At the end of this step, a robust estimate of the rotation of the object is obtained. However, this does not involve information from keypoint detectors, as they are not found to be reliable enough.

Consensus: This step involves identifying and removing outlier points. To do this, hierarchical agglomerative clustering is applied on the set of votes V based on the Euclidean distance as the dissimilarity measure.

To summarize, this technique is a novel keypoint based method for long term model free tracking where a consensus based scheme is used for outlier detection and a voting mechanism is used to determine object centre. This method has the advantage that it is highly accurate and it can achieve consistency in tracking over a large number of frames. However, it suffers from the drawback of large computational complexity especially when robust keypoint detectors like BRISK, SIFT or SURF are used.

2.2.3. Conventional Feature Detection and Matching

First, a set of feature points is found in a frame. This is followed by a pyramidal Lucas-Kanade Algorithm that is used to track the same points in the next image. Of the set of points in two images that are almost matched, some of the points are on moving objects but most are on background. To remove the effect of the moving background, several steps are computed. First, a robust least-squares algorithm, RANSAC[36] is used to find an affine matrix to find a mapping from one image to the other. The affine matrix is used to warp the second image and then the warped image is subtracted from the first image. The background disappears because the feature points remaining after the robust least-squares are almost all on the background. What show up are the objects that are moving relative to the background. Then, a connected components algorithm is used to group pixels into object groups resulting in a moving object blob detector [27]. Finally, a basic Kalman filter tracker with a constant acceleration motion model is used to track the objects. Prediction is done based on the target position and velocity, and a global nearest neighbor (GNN) algorithm is used to pick the most likely object for the track in the neighborhood of the point.

2.3 OBJECT TRACKING METHODS NOT BASED ON MATCHING FOLLOWED BY TRACKING

Two of the object-tracking methods which are based on a completely different flow are:

1. Tracking via Sparse Representation [15]
2. Tracking with Online Multiple Instance Learning [16]

2.3.1 Tracking via Sparse Representation [15]:

This method approaches tracking as a sparse approximation problem in a particle filter framework. In this framework, occlusion, noise, and other challenging issues are

addressed seamlessly through a set of trivial templates. Specifically, to find the tracking target in a new frame, each target candidate is sparsely represented in the space spanned by target templates and trivial templates. The sparsity is achieved by solving an l_1-regularized least-squares problem. Then, the candidate with the smallest projection error is taken as the tracking target. After that, tracking is continued using a Bayesian state inference framework. The tracking performance is further improved by using two strategies. First, target templates are dynamically updated to capture appearance changes. Second, non-negativity constraints are enforced to filter out clutter which negatively resembles tracking targets.

2.3.2 Tracking with Online Multiple Instance learning [16]:

In this approach, the problem of tracking an object in a video given its location in the first frame and no other information is addressed. A class of tracking techniques called "tracking by detection" has been shown to give promising results at real-time speeds. These methods train a discriminative classifier in an online manner to separate the object from the background. This classifier bootstraps itself by using the current tracker state to extract positive and negative examples from the current frame. Slight inaccuracies in the tracker can therefore lead to incorrectly labeled training examples, which degrade the classifier and can cause drift. Using Multiple Instance Learning (MIL) instead of traditional supervised learning avoids these problems and can lead to a more robust tracker with fewer parameter tweaks. A new online MIL algorithm for object tracking that achieves superior results with real-time performance is used in this approach.

2.4 COMPARISON OF THE DIFFERENT TECHNIQUES FOR OBJECT TRACKING

In the method involving mean shift with SIFT features, mean shift is a non-parametric feature-space analysis technique for locating the maxima of a density function, a so-called mode-seeking algorithm. When used for clustering, it is robust to image occlusions and clutters. Also, it is capable of handling arbitrary feature shapes and data cluster shapes. However, it suffers from two main drawbacks. First, it is computationally intensive and requires $O(kN^2)$ operations, where N is the number of data points and k is the number of average iteration steps for each data point. Second, the mean shift algorithm relies on sufficient high data density with clear gradient to locate the cluster centers. In particular, the mean shift algorithm often fails to find appropriate clusters for so called data outliers, or those data points located between natural clusters. Though SIFT is the most widely used feature detection algorithm due to its accuracy and invariance to rotation and scale, it is computationally expensive due to the high dimensionality of the descriptor at the matching step. Also, the similarity measures for correspondence between two groups of the sample points demands two sequential operations for the pdf and integral calculations that are of the order of $O(N^2)$. This makes it less attractive for parallel implementations as execution time is high.

Consensus based matching and tracking of keypoints has high accuracy when compared to the existing approaches in model-free object tracking. It has been shown that it achieves state-of-the-art results over a large number of sequences. However this approach requires manual initialization over the first frame and then carrying out the tracking in subsequent frames which are not exactly the approach we are interested in. Also, this

method involves keypoint detection which is computationally expensive when prominent keypoint detection techniques such as BRISK, SIFT or SURF are used.

Tracking with sparse representation requires a lot of memory to store the templates. Also, a Bayesian inference network is very complex to be implemented on an accelerator. Tracking with multiple instance learning involves using multiple instance learning in training a classifier which is not the current focus of the approach we are interested in. These drawbacks make these methods unattractive for implementation on the hardware accelerator.

CHAPTER 3

PROPOSED OBJECT TRACKING FRAMEWORK

3.1 MOTIVATION

The complexity of the existing techniques, namely, mean shift, consensus based tracking, tracking with sparse representation and tracking with multiple instance learning is quite high. In this thesis, we focus on developing a low cost object tracking system. Consequently, the algorithms that were not computationally intensive but achieved good performance were chosen.

3.2 OVERVIEW

The proposed object tracking system is based on the technique described in [3] The task of object tracking is divided into three steps:

Step 1: Foreground detection: Identifying the moving objects in a video.

Step 2: Clustering: Representing each moving object by its centroid.

Step 3: Tracking: Estimating the parameters of motion of each object

Our proposed technique performs foreground detection using an improved version of adaptive Gaussian Mixture model developed by Zivkovic [4]. This is followed by the proposed clustering algorithm that uses DBSCAN clustering algorithm on a frame followed by K-means clustering for 9 frames and a diagnostic logic to figure out the right number of clusters from K-means. The last step uses the Kalman filter to predict estimates of the object's position and velocity. The block diagram of the whole system is summarized in Figure 1:

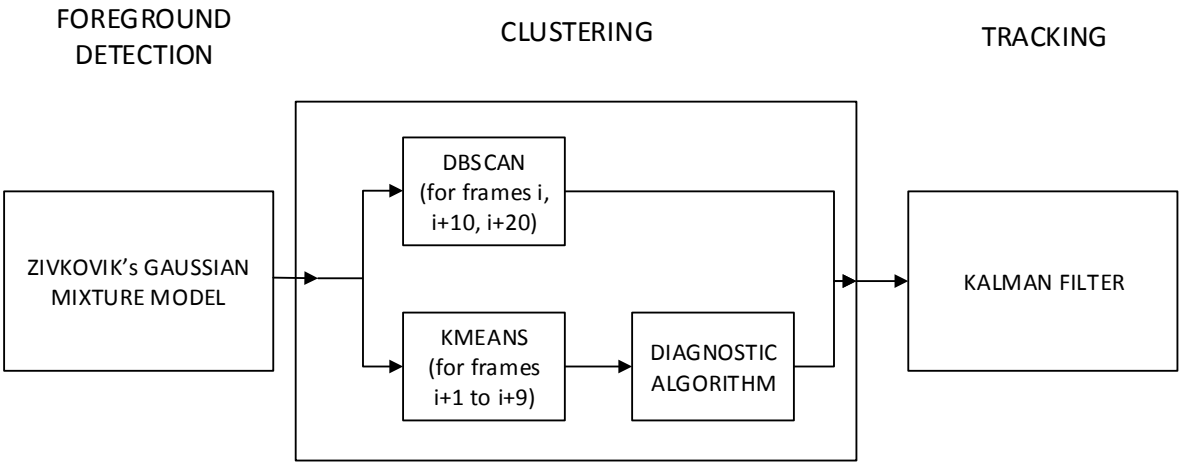


Figure 1. Proposed Object tracking system

3.3 DETAILED DESCRIPTION

3.3.1 Foreground Detection

The foreground detection is done using an improved version of adaptive Gaussian Mixture Model (GMM) developed by Zivkovic [4]. This is an efficient adaptive algorithm using Gaussian mixture probability density. Recursive equations are used to update the parameters and also to simultaneously select the appropriate number of components for each pixel. An advantage of using Gaussian mixture model is that it is robust to changes in illumination. It adapts to changes by updating the training set with new samples and discarding old samples from the training set. We choose a reasonable time period *T*, and at time *t*, we have training data set, $\chi_T = \{x^{(t)}, \ldots, x^{(t-T)}\}$ where $x^{(t)}$ is the value of the pixel at time t . For each new sample we update the training data set χ_T and re-estimate $\hat{p}(\vec{x}|\chi_T, BG)$. However, among the samples from the recent history there

could be some values that belong to the foreground objects and we should denote this estimate as $\hat{p}\,(\vec{x}^{(t)}|\chi_T, BG+FG)$. We use GMM with M components:

$$\hat{p}(\vec{x}|\chi_T, BG+FG) = \sum_{m=1}^{M} \hat{\pi}_m N\,(\vec{x}; \hat{\vec{\mu}}_m, \hat{\sigma}_m{}^2 I) \quad (1)$$

where $\hat{\vec{\mu}}_{1,\ldots,}\,\hat{\vec{\mu}}_m$ are the estimates of the means and $\hat{\sigma}_1,\ldots.\hat{\sigma}_m$ are the estimates of the variances that describe the Gaussian components. The mixing weights denoted by $\hat{\pi}_m$ are non-negative and add up to one. The covariance matrices are assumed to be diagonal and the identity matrix I has proper dimensions. Given a new data sample $\vec{x}^{(t)}$ at time t the recursive update equations are [5]

$$\hat{\pi}_m \leftarrow \hat{\pi}_m + \alpha(o_m^{(t)} - \hat{\pi}_m) \quad (2)$$

$$\hat{\vec{\mu}}_m \leftarrow \hat{\vec{\mu}}_m + o_m^{(t)}(\alpha/\hat{\pi}_m)\vec{\delta}_m \quad (3)$$

$$\hat{\sigma}_m^2 \leftarrow \hat{\sigma}_m^2 + o_m^{(t)}(\alpha/\hat{\pi}_m)\,(\vec{\delta}_m^T\,\vec{\delta}_m - \hat{\sigma}_m^2)\,, \quad (4)$$

where $\vec{\delta}_m = \vec{x}^{(t)} - \hat{\vec{\mu}}_m$. Instead of the time interval T that was mentioned above, here constant α describes an exponentially decaying envelope that is used to limit the influence of the old data. We keep the same notation having in mind that α is approximately 1/T. For a new sample the ownership $o_m^{(t)}$ is set to 1 for the 'close' component with largest $\hat{\pi}_m$ and the others are set to zero. We define that a sample is 'close' to a component if the Mahalanobis distance from the component is for example less than three standard deviations. The squared distance from the m-th component is calculated as: $D_m^2(\vec{x}^{(t)}) = \vec{\delta}_m^T\,\vec{\delta}_m/\hat{\sigma}_m^2$. If there are no 'close components' a new component is generated with $\hat{\pi}_{m+1} = \alpha$, $\hat{\vec{\mu}}_{m+1} = \vec{x}^{(t)}$ and $\hat{\sigma}_{m+1} = \hat{\sigma}_0$ where $\hat{\sigma}_0$ is some appropriate initial variance. If the maximum number of components is reached, we discard the component with smallest $\hat{\pi}_m$.

In this algorithm, we approximate the background model by the first B largest clusters:

$$p(\vec{x}|X_T, BG) \sim \sum_{m=1}^{B} \hat{\pi}_m N(\vec{x}; \hat{\vec{\mu}}_m, \hat{\sigma}_m^2 I) \quad (5)$$

If the components are sorted to have descending weights $\hat{\pi}_m$, we have

$$B = \arg min\left(\sum_{m=1}^{M} \hat{\pi}_m > (1-c_f)\right), \quad (6)$$

where c_f is a measure of the maximum portion of the data that can belong to foreground objects without influencing the background model. For example, if a new object comes into a scene and remains static for some time, it will probably generate an additional stable cluster. Since the old background is occluded, the weight π_{B+1} of the new cluster will be constantly increasing. If the object remains static long enough, its weight becomes larger than c_f and it can be considered to be part of the background. If we look at (2) we can conclude that the object should be static for approximately $\log(1 - c_f)/\log(1 - \alpha)$ frames. For example for $c_f = 0.1$ and $\alpha = 0.001$ we get 105 frames.

The procedure for selecting the number of components is as follows: The weight π_m describes how much of the data belongs to the *m*-th component of the GMM. It can be regarded as the probability that a sample comes from the *m*-th component and in this way the π_m-s define an underlying multinomial distribution. Let us assume that we have *t* data samples and each of them belongs to one of the components of the GMM. Let us also assume that the number of samples that belong to the *m*-th component is $n_m = \sum_{i=1}^{t} o_m^{(i)}$ where $o_m^{(i)}$-s are the ownerships. The assumed multinomial distribution for n_m-s gives likelihood function $\mathcal{L}=\prod_{m=1}^{M} \pi_m^{n_m}$. The mixing weights are constrained to sum up to one. We take this into account by introducing the Lagrange multiplier λ. The Maximum Likelihood (ML) estimate follows from: $\frac{\partial}{\partial \hat{\pi}_m}(log\ L + \lambda(\sum_{m=1}^{M} \hat{\pi}_m - 1)) = 0$. After getting rid of λ we get:

$$\hat{\pi}_m^{(t)} = \frac{n_m}{t} = \frac{1}{t}\sum_{i=1}^{t} o_m^{(i)} \quad (7)$$

The estimate from *t* samples, denoted as $\hat{\pi}_m^{(t)}$ and it can be rewritten in recursive form as a function of the estimate $\hat{\pi}_m^{(t-1)}$ for *t*-1 samples and the ownership $o_m^{(i)}$ of the last sample:

$$\hat{\pi}_m^{(t)} = \hat{\pi}_m^{(t-1)} + 1/t(o_m^{(t)} - \hat{\pi}_m^{(t-1)}) \quad (8)$$

If we now fix the influence of the new samples by fixing 1/t to α = 1/T we get the update equation (2). This fixed influence of the new samples means that we rely more on the new samples and the contribution from the old samples is down weighted in an exponentially decaying manner as mentioned before.

Prior knowledge for multinomial distribution can be introduced by using its conjugate prior, the Dirichlet prior $\mathcal{P} = \prod_{m=1}^{M} \pi_m^{c_m}$. The coefficients c_m have a meaningful interpretation. For the multinomial distribution, the c_m presents the prior evidence (in the maximum aposteriori (MAP) sense) for the class *m* - the number of samples that belong to that class a priori. As in [5] we use negative coefficients c_m = -c. Negative prior evidence means that we will accept that the class *m* exists only if there is enough evidence from the data for the existence of this class. This type of prior is also related to Minimum Message Length criterion that is used for selecting proper models for given data [5]. The MAP solution that includes the mentioned prior follows from $\frac{\partial}{\partial \hat{\pi}_m}(log\ L + log\ \mathcal{P} + \lambda(\sum_{m=1}^{M} \hat{\pi}_m - 1)) = 0$, where $\mathcal{P} = \prod_{m=1}^{M} \pi_m^{-c}$. We get:

$$\hat{\pi}_m^{(t)} = \frac{1}{\mathrm{K}}(\sum_{i=1}^{t} o_m^{(i)} - c), \quad (9)$$

where $\mathrm{K} = \sum_{m=1}^{M}(\sum_{i=1}^{t} o_m^{(i)} - c) = t\text{-}Mc$. We rewrite (9) as:

$$\hat{\pi}_m^{(t)} = \frac{\hat{\Pi}_m^{(t)} - c/t}{1 - Mc/t} \quad (10)$$

where $\widehat{\Pi}m = \frac{1}{t}\sum_{i=1}^{t} o_m^{(i)}$ is the ML estimate from (7) and the bias from the prior is introduced through *c/t*. The bias decreases for larger data sets (larger t). However, if a small bias is acceptable we can keep it constant by fixing c/t to $c_T = c/T$ with some large *T*. This means that the bias will always be the same as if it would have been for a data set with *T* samples. It is easy to show that the recursive version of (9) with fixed $c/t = c_T$ is given by:

$$\hat{\pi}_m^{(t)} = \hat{\pi}_m^{(t-1)} + 1/t\left(\frac{o_m^{(t)}}{1-Mc_T} - \hat{\pi}_m^{(t-1)}\right) - 1/t\frac{c_T}{1-Mc_T} \quad (11)$$

Since we expect usually only a few components M and c_T is small we assume $1\text{-}Mc_T \approx 1$. As mentioned we set $1/t$ to α and get the final modified adaptive update equation

$$\hat{\pi}_m \leftarrow \hat{\pi}_m + \alpha\left(o_m^{(t)} - \hat{\pi}_m\right) - \alpha c_T \quad (12)$$

The above equation is used instead of (2). After each update we need to normalize $\hat{\pi}_m$-s so that they add up to one. We start with GMM with one component centered on the first sample and new components are added as mentioned in the previous section. The Dirichlet prior with negative weights will suppress the components that are not supported by the data and we discard the component *m* when its weight π_m becomes negative. This also ensures that the mixing weights stay non-negative. For a chosen $\alpha = 1/T$ we could require that at least $c = 0.01 * T$ samples support a component and we get $c_T = 0.01$.

3.3.2 Clustering

Clustering is a data mining technique that groups data into meaningful subclasses, known as clusters, such that it minimizes the intra-differences and maximizes inter-differences of these subclasses [8]. Well-known algorithms include K-means, K-medoids [37], BIRCH [38], DBSCAN, STING [39], and WaveCluster [40]. These algorithms

have been used in various scientific areas such as satellite image segmentation, noise filtering and outlier detection, unsupervised document clustering, and clustering of bioinformatics data. DBSCAN is accurate but computationally intensive. K-means, on the other hand, is fast but requires the number of clusters as an input while also being prone to noise. We propose an algorithm that uses DBSCAN for a single frame out of a group of M frames followed by K-means clustering for the remaining M-1 frames. For K-means, we apply a diagnostic algorithm to estimate the number of means that is closest to the actual number of means before launching it. The details of this are explained in subsequent sections.

DBSCAN Algorithm: DBSCAN (Density-Based Spatial Clustering of Applications with Noise), introduced by Ester et al [9], is a non-parametric, density based clustering technique [7]. It assumes that cluster is a region in the data space with high density. Compared to non-density based clustering methods, the DBSCAN algorithm has unique and advanced features that are useful when detecting objects/class/patterns/structures of different shapes and sizes. DBSCAN is a good candidate to find 'natural' clusters and their arrangement within the data space when they have a comparable density without any preliminary information about the groups present in a data set.

DBSCAN scans the points in the input, one at a time, and grows clusters around each point if it is able to find sufficient number of points within a certain neighborhood of the point considered. The two inputs to the algorithm are *Eps*, which is a distance metric that defines the neighborhood of a point (this is simply a sphere around the point of radius Eps if we are considering Euclidean distance), and *MinPts*, which is the minimum

number of points required to be in the Eps neighborhood of a point that would allow a cluster to be grown from.

To find a cluster, DBSCAN starts with an arbitrary point *p* and retrieves all points reachable from *p* with respect to *Eps* and *MinPts*. If *p* is a core point (i.e. a point that has at least *MinPts* points in its *Eps* neighborhood), this procedure yields a cluster with respect to *Eps* and *MinPts*. If *p* is not a core point, then DBSCA N moves onto the next point.

The pseudo code for the algorithm as published in the original nomenclature [9] is as follows:

```
DBSCAN(D, eps, MinPts) {
   C = 0
   for each point P in dataset D {
      if P is visited
         continue next point
      mark P as visited
      NeighborPts = regionQuery(P, eps)
      if sizeof(NeighborPts) < MinPts
         mark P as NOISE
      else {
         C = next cluster
         expandCluster(P, NeighborPts, C, eps, MinPts)
      }
   }
}


expandCluster(P, NeighborPts, C, eps, MinPts) {
   add P to cluster C
   for each point P' in NeighborPts {
      if P' is not visited {
         mark P' as visited
         NeighborPts' = regionQuery(P', eps)
         if sizeof(NeighborPts') >= MinPts
            NeighborPts = NeighborPts joined with NeighborPts'
      }
      if P' is not yet member of any cluster
         add P' to cluster C
   }
}
```

```
regionQuery(P, eps)
   return all points within P's eps-neighborhood (including
P)
```

Figure 2. DBSCAN Algorithm

K-means clustering [12]: K-means is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. It aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

Description: Given a set of observations ($\mathbf{x}_1$, $\mathbf{x}_2$, …, $\mathbf{x}_n$), where each observation is a *d*-dimensional real vector, *K-means* clustering aims to partition the *n* observations into k ($\leq n$) sets $\mathbf{S} = \{S_1, S_2, \ldots, S_k\}$ so as to minimize the within-cluster sum of squares (WCSS). In other words, its objective is to find:

$$\sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2, \tag{13}$$

where $\boldsymbol{\mu}_i$ is the mean of points in S_i.

Algorithm: The most common algorithm uses an iterative refinement technique. It is also called Lloyd's algorithm. Given an initial set of *K-means* $m_1^{(1)},\ldots,m_k^{(1)}$, the algorithm proceeds by alternating between two steps:

Assignment step: Assign each observation to the cluster whose mean yields the least within-cluster sum of squares (WCSS). Since the sum of squares is the squared Euclidean distance, this is intuitively the "nearest" mean.

$$S_i^{(t)} = \{\mathrm{x_p} : \left\|x_p - m_i^{(t)}\right\|^2 \leq \left\|x_p - m_j^{(t)}\right\|^2 \ \forall j, 1 \leq j \leq k\}, \tag{14}$$

where each x_p is assigned to exactly one $S^{(t)}$, even if it could be assigned to two or more of them.

Update step: Calculate the new means to be the centroids of the observations in the new clusters.

$$m_i^{(t+1)} = \frac{1}{\left|S_i^{(t)}\right|} \sum_{x_j \in S_i^{(t)}} x_j \tag{15}$$

Since the arithmetic mean is a least-squares estimator, this also minimizes the within-cluster sum of squares (WCSS) objective. The algorithm converges when the assignments no longer change.

Comparison of K-means and DBSCAN clustering:

DBSCAN is a very robust algorithm since it is good at finding out arbitrarily shaped clusters. It is robust to outliers since it has inherent noise rejection capabilities. Also, it requires just two parameters and is insensitive to the ordering of the points in the database. One does not need to specify the number of means beforehand for it, unlike K-means. However, DBSCAN suffers from the drawback that it is not entirely deterministic and border points that are reachable from more than one cluster can be part of either cluster, depending on the order the data is processed. Also, the quality of DBSCAN depends on the distance measure used in the function regionQuery(P,ε). The most common distance metric used is Euclidean distance. But this metric becomes useless for high dimensional data due to the curse of dimensionality. Also, since the order of complexity is $O(N^2)$ where N is the number of pixels, DBSCAN gets computationally intensive as the size of input data increases.

K-means is a computationally simple algorithm with order of complexity as $O(k\mathrm{N})$, where k is the number of clusters. The metric here is Euclidean distance. Its main drawback is that it needs to know the number of clusters k beforehand. An inappropriate choice of k can lead to poor results. Also, the final means returned by K-means depends on the initial set of means. This could result in poor accuracy at times. But it is computationally very simple for lower dimensional data which makes it attractive for low cost implementations. The problems of both DBSCAN and K-means algorithms can be overcome by simply combining them together with additional modifications.

3.3.3 Proposed Clustering Algorithm

In the proposed method, DBSCAN clustering is performed once every M frames and K-means clustering is performed in the remaining M-1 frames since K-means is way less computationally expensive. We experimented with different values of M and found that M=10 results in a significant decrease in runtime without losing much accuracy. The results are presented in Section 4.4

The number of centroids obtained by DBSCAN algorithm is sent to K-means. The K-means operates on N-2 to N+2 centroids, where N denotes the number of centroids in the previous frame. The value of N for the current frame is determined through a diagnostic algorithm that rejects noise, but accepts new objects that may have entered the scene. The diagnostic algorithm works using distance as metric. It filters out the noise points using a size threshold. The proposed algorithm is summarized below:

1. In the first frame (out of a group of M frames), perform DBSCAN. Let the number of centroids returned be N.

2. In the next frame, run K clustering algorithm from N-X to N+X, where X is a variable that can take values 2, 3, 4 or 5. The value of X is different for different videos and is determined experimentally.

3. Store the number of centroids obtained for each iteration and also the size of the clusters. Calculate the distance between the centroids of the current frame to the centroids of the previous frame. We create a distance matrix which would be of dimensions AxB, where A corresponds to the number of centroids in the current frame and B corresponds to the number of centroids in the previous frame. A matrix element with index (a,b) would correspond to the distance between a^{th} centroid in the current frame and the b^{th} centroid in the previous frame. We establish a correspondence with the points in the previous frame such that the distance values are the minimum and no two points in the current frame correspond to the same point in the previous frame. We ensure this by looking at the value of 'b^{th}' co-ordinate. The number of means such that the distances obtained with the corresponding centroids in the previous frame are the minimum is the right number of means. Filter out the noise points after this based on a size threshold, the value of which is determined by repeated experiments.

4. We repeat the above procedure for the subsequent frames, with two modifications: (i) the number of means N is taken from the previous frame and (ii) instead of minimum distance, we pick the number of centroids as the one who's distances from the centroids of the previous frame lies within a range of distances returned in the previous iteration.

A pseudo code for our proposed algorithm is presented below:

```
## Main
centroids = DBSCAN(frame[0])
Foreach centroid {
  new Object (new_name, centroid)
  push current_objects(frame(0)), Object
}
Cluster(frames)
## End Main

AssociateObjects(centroids, old_objects) {
  New_objects = null
  D = 0
  foreach obj in old_objects {
      cen1 = obj.centroid
      foreach cen2 in centroids {
        dist_matrix[obj][cen2] = distance(cen1, cen2)
      }
   }
   foreach cen2 in centroids {
      Next if cen2.cluster_size < size_threshold
      closest_object = obj for which dist_matrix[obj][cen2] is
minimum
      if dist_matrix[obj][cen2] < distance_threshold
          push new_objects, obj
          obj.update_centroid(cen2)
      else
          new_obj = New Object(cen2)
          push new_objects, new_obj
   }
   return new_objects
}

Cluster(Frames) {
   B = 0
   I = 0
   While B <= Total number of frames {
      centroids = DBSCAN(frame[I+B])
      Objects(frame(I+B)), weight=AssociateObjects(centroids,
curr_objects)
      while I < 10 {
        curr_objects = Objects(frame(I+B)
        N = count (centroids)
        I++
         Weight = inf
        For N - 2 <= J <= N + 2 {
          ## J is number of clusters for K-means
          K[J]_centroids = K-means(frame[I+B], J)
```

```
          temp_objects,
temp_weight=AssociateObjects(K[J]_centroids, curr_objects)
          if temp_weight < weight
             centroids = K[J]_centroids
              next_frame_objects = temp_objects
        }
        Objects(frame(I+B)) = next_frame_objects
      }
    }
}
```

Figure 3. Proposed Clustering Algorithm

3.3.4 Kalman Filter [13]

A Kalman filter is an optimal recursive data processing algorithm. It processes all available measurements, regardless of their precision, to estimate the current value of the variables of interest, with use of (1) knowledge of the system and measurement device dynamics, (2) the statistical description of the system noise, measurement errors, and uncertainty in the dynamics models, and (3) any available information about initial conditions of the variables of interest. Since a Kalman filter is recursive, there is no need to require all previous data to be kept in storage and reprocessed every time a new measurement is taken.

A Kalman filter combines all available measurement data, plus prior knowledge about the system and measuring devices, to produce an estimate of the desired variables in such a manner that the error is minimized statistically. It estimates a process by using a form of feedback control: the filter estimates the process state at some time and then obtains feedback in the form of (noisy) measurements. As such, the equations for the Kalman filter fall into two groups: time update equations and measurement update equations. The time update equations are responsible for projecting forward (in time) the

current state and error covariance estimates to obtain the a priori estimates for the next time step. The measurement update equations are responsible for the feedback, i.e. for incorporating a new measurement into the a priori estimate to obtain an improved a posteriori estimate. The equations are updated as shown in Figure 6.

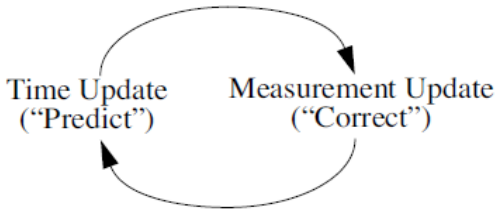


Figure 4: The discrete Kalman filter cycle

Discrete Kalman filter time update equations:

Project the state ahead: $\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k$ (17)

Project the error covariance ahead: $P_k^- = AP_{k-1}A^T + Q$ (18)

Discrete Kalman filter measurement update equations:

Compute the Kalman gain: $K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$ (19)

Update the error covariance: $P_k = (I - K_k H)P_k^-$ (20)

Update estimate with measurement z_k: $\hat{x}_k = \hat{x}_k^- + K_k (z_k - H\hat{x}_k^-)$ (21)

The variables used in eqns (17) – (21) are described below:

$\hat{x}_k^-$: The a posteriori state estimate at time k given observations up to and including at time k. It is represented by $\begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix}$, where (x,y) is the co-ordinate of the point and v_x, v_y are the velocities along x and y directions respectively.

A : State transition model which is applied to the previous state $\mathbf{x}_{k-1}$

The value used in this work is: $\begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

B : Control input model which is applied to control vector u_k . Here a zero matrix.

P_k : The *a posteriori* error covariance matrix

H : The observation model which maps the true state space into the observed space.

This matrix is $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$

z_k : Observation at time instant 'k'

The first task during the measurement update is to compute the Kalman gain, K_k. The next step is to actually measure the process to obtain, and then to generate an *a posteriori* state estimate by incorporating the measurements in equation (20). The final step is to obtain an *a posteriori* error covariance estimate via equation (21). After each time and measurement update pair, the process is repeated with the previous *aposteriori* estimates used to project or predict the new *apriori* estimates.

In addition, there is some form of a measurement model [14] that describes the relationship between the process state and measurements. This can usually be represented with a linear expression similar to equation:

$$z_k = Hx_k + v_k \tag{22}$$

where, z_k is a 2x1 matrix with elements $z_k[1]$ is the x coordinate measurement obtained directly from pixels, $z_k[2]$ is the y coordinate measurement obtained directly from pixels, and v_k is a 2x1 measurement noise vector with covariance matrix R. that is also a 2x2 matrix. R is the measurement noise covariance of size 2x2. It determines how much

information from the measurement is used [41]. So, if R is high, the Kalman Filter considers the measurements as not very accurate. In our work, R is obtained from the covariance measurements done on the image pixels.

CHAPTER 4

RESULTS

4.1 SIMULATION FRAMEWORK

The proposed object tracking system was tested on video sequences of cars that were captured using stationary cameras. The videos were taken from www.changedetection.net [17]. Since the simulation setup currently does not have support for OpenCV's video subroutines, only offline processing could be carried out. The video frames were extracted and stored for offline processing. The sizes of the video frames were 320x240, 320x240 and 308x242. They were all 8-bit grayscale images.

The first video sequence, labeled Aerial_highway, is a video sequence that captures the traffic at a junction of two crossroads from a stationary aerial camera. The sequence of frames has 5 moving cars. The second video, labeled Multi_car, is the video captured from a road with its sidewalk. This has cars moving on the road and a bicycle on the pavement. The third frame sequence, labeled Highway, has four moving objects. The considered frame sequences have objects moving as well as ones which are entering the scene.

The proposed algorithm was implemented in C++. Sequential codes for DBSCAN and Zivkovic's Gaussian Mixture Model were obtained from Open Source Software [4]. The accelerator was simulated using Gem5 [33]. Since the GEM5 simulations on the accelerator were very slow, we tested the algorithm for fifteen frames of each video sequence.

For each of the video sequences, we checked the speedups of the individual kernels. The speedup was the ratio of the reference code execution time and our implementation of the kernel.

For example, for GMM, speedup was calculated as the ratio of execution time for sequential implementation to the execution time for parallel implementation. For clustering, the speedup is calculated as the ratio of the execution time for DBSCAN algorithm to the execution time for the proposed clustering algorithm.

4.2 DESCRIPTION OF THE HARDWARE ACCELERATOR

The platform used to implement the algorithm is a warp-based SIMT coprocessor. There are 8 virtual warps; each warp consists of 32 threads that operate on 32 lanes of data simultaneously. The lanes each have their own integer register file, a separate floating point register file, and conditional flags. The programmer sees a warp of 32 lanes that can execute integer operations and single and double precision floating point (FP) operations. The processor can switch between 8 virtual warps, similar to nVIDIA Fermi GPU.

The instruction set supported is ARM's 64-bit AArch ISA. The implementation is embodied in the gem5 simulator. The massively parallel engine achieves high throughput by employing extreme multi threading. Currently, upto 256 threads are supported. To hold the context of these threads, this engine supports a register file of around 600KB size.

4.3 PARALLELIZATION SCHEMES

Gaussian Filter:

To find the most suitable parallelization scheme for the hardware accelerator, we first carried out experiments involving 2D Gaussian filter of size 5x5. Here 1D filter computations were done along the columns followed by 1D filter computations along the rows. Input image is a 2-D matrix of unsigned int data type with 16bits/pixel. The coefficients are 16 bit integers. 3 types of input images are used: Small(512x512), Medium(1024x1024) and Large(2048x2048) .

Different parallelization schemes were experimented with, namely, pixel-level parallelization, loop unrolling along rows and loop unrolling along columns. In pixel level parallelization, the number of threads launched is equal to the total number of pixels in the image and the computations on individual pixels are carried out in parallel. The order in which the threads are spawned and managed is decided by the compiler. In the loop unrolling approach across columns, the number of threads spawned is equal to the number of columns. Each thread traverses down the column in steps of the loop unrolling factor. By this, we mean that in each thread, there are two loops. The inner loop works on generating N outputs at a time, where N is the loop unrolling factor. The outer loop traverses down the column in blocks of N pixels (Figure 8). In case of loop unrolling along rows, the same approach is taken by spawning one thread per row and traversing across the row in each thread.

For the Gaussian filter, the loop unrolling approach along columns had the best speedups among the schemes that were analyzed. This was because it exploited the characteristics of cache-line access the best. Pixels adjacent to each other along a row are

stored in the same cache-line. As a result, there are more load reuses compared to the loop unrolling approach along the rows. In pixel level parallelization, each pixel is processed in a different thread. While this also provides opportunity for exploiting cache line access depending upon the order in which the threads are executed, there is an overhead in managing the large number of threads.

We analyzed the variation in performance with loop unrolling factor for Gaussian column filters and the results are presented in Figure 9. Our study shows that a loop unrolling factor of eleven along the column achieves the best performance on the accelerator; larger unrolling factors have minimal effect on the performance.

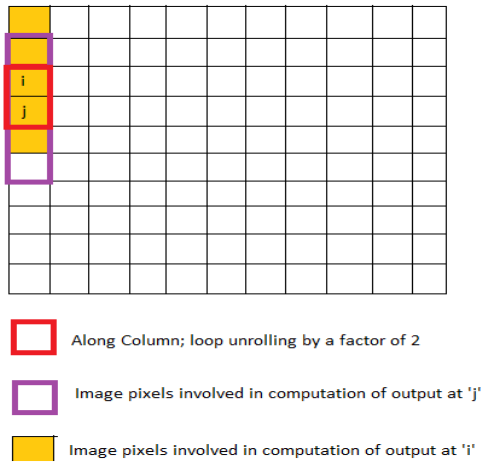


Figure 5: Loop unrolling by a factor of 2 implies 2 outputs per traversal down the thread; N threads

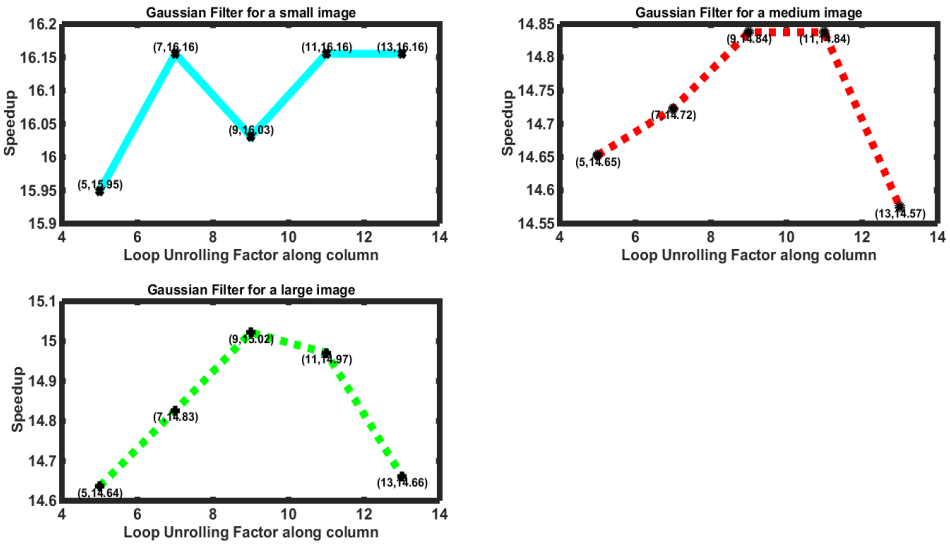


Figure 6: Variation in speedups for Gaussian filter with loop unrolling factor; best performance is when column loop unrolling factor is 11 for small and medium images and 9 for large images

Gaussian Mixture Model:

For Gaussian Mixture Model (GMM), we investigated the performance of pixel level parallelization and loop unrolling. However, owing to prohibitively long simulation times on the GEM5 for GMM, we considered loop unrolling factors of 1, 2, 4 and 8. Speedup is typically a function of the load store reuses and exploiting cache-line accesses. So while the optimal loop unrolling factor for Gaussian filter was found out to be 11, for the GMM, if we chose a factor that is not a divisor of the height of the image, the overhead caused due to the additional computations for the remaining pixels of the column was significant. We found that the speedup was almost the same across all loop unrolling factors. The results are shown in Figure 7.

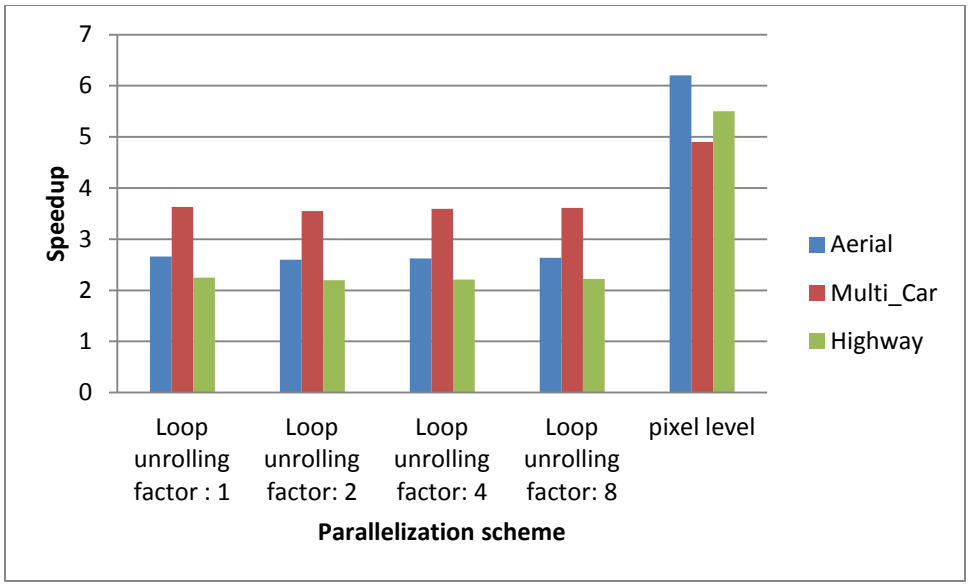


Figure 7: Speedup for foreground detection using different parallelization schemes

The speedups for foreground detection on the hardware accelerator using different parallelization schemes are shown in Figure 7. As we can see, these results indicate that pixel level parallelization shows the best performance. There is no big difference in the speedup performance of different loop unrolling factors because the serial execution time and parallel execution time changed proportionately; the variation was in few thousands of microseconds that made up 0.1% of the total execution time.

DBSCAN:

The sequential DBSCAN algorithm scans the points in space sequentially and grows clusters around the current point, marking the points to which it grows as visited. It then moves on to unvisited points and repeats the same procedure. Attempts have been made to parallelize this algorithm [8], however the parallelization requires additional runtime for book-keeping tasks and merging of threads. In fact, the more the number of cores, the more time the algorithm takes for merging the threads. The memory requirements of the merging operation were too large for the hardware accelerator and

simulations terminated without any errors or print statements. As a result, we were not able to implement a parallel version of DBSCAN. Instead we designed a low cost algorithm that exploits the fact that the objects to be tracked do not change much between subsequent frames and hence the locations of the centroids are also close to each other.

4.4 TRACKING PERFORMANCE

To evaluate the accuracy of our results, outputs were inspected at the end of every stage. For foreground detection, we checked the output by reading the text file output from GEM5 and visually checking it in MATLAB. We used the DBSCAN as the baseline for our clustering algorithm and compared the performance of our proposed algorithm with DBSCAN. The centroids returned by our algorithm was the same as those returned by DBSCAN in most cases. However, there were a few deviations due to noise but those errors were rectified at the Kalman filter stage.

In order to determine the effect of the number of clusters N, on the accuracy of clustering (i.e. tracking performance), we compared the results of our clustering approach by computing the error between the centroids returned per frame with our proposed algorithm and the centroids returned if DBSCAN had been done on each frame. The error was calculated by taking a root mean square of the Euclidean distance between each centroid returned by DBSCAN, and the closest centroid returned by the proposed solution for that frame. We conducted an experiment in which we fixed the frame parameter M and varied the number of means over which K means was iterated (N-2 to N+2, N-3 to N+3, N-4 to N+4 and N-5 to N+5). The frame parameter is number of frames over which K-means and DBSCAN is repeated such that DBSCAN is done for 1

frame and K-means is done for M-1 frames. We varied the frame parameter as 10 and 12, since these gave the best results for the three videos. Figures 8 and 9 show the RMS error for 210 frames of multi_car video for frame parameters 10 and 12; Figures 10 and 11 for 210 frames of highway video sequence; Figures 12 and 13 for 210 frames of Aerial_highway video. We found that the frame parameter and number of means that gave the best result is different for different video sequences. The findings have been presented in Table 1.

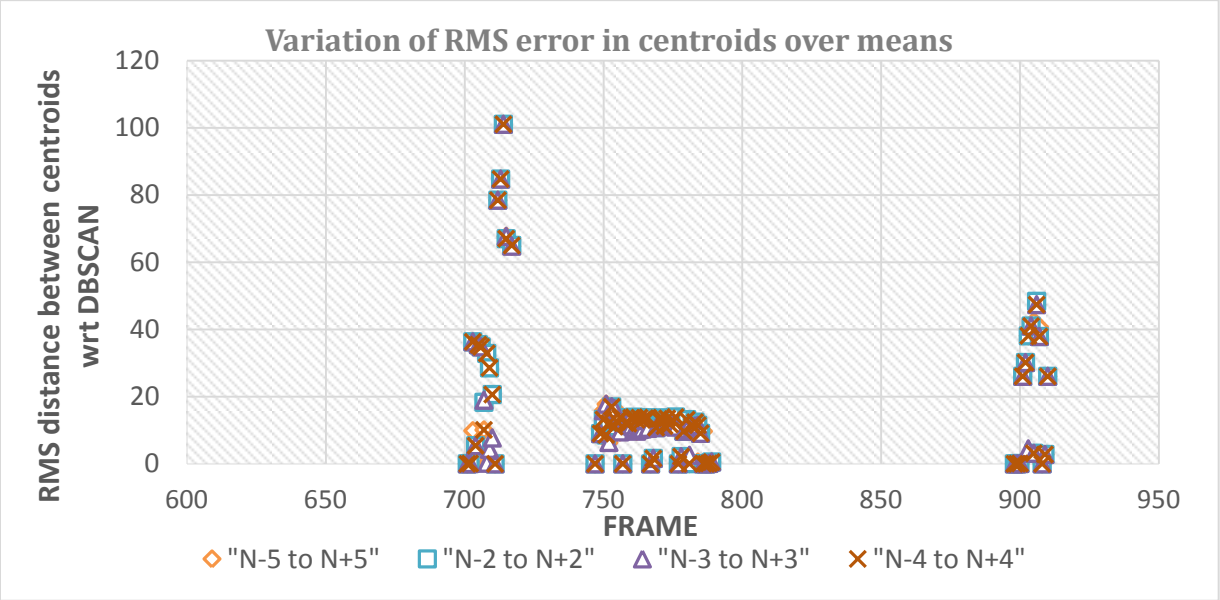


Figure 8: Variation in tracking performance with the number of frames over which K Means is done for multi_car video. Frame parameter is 10.

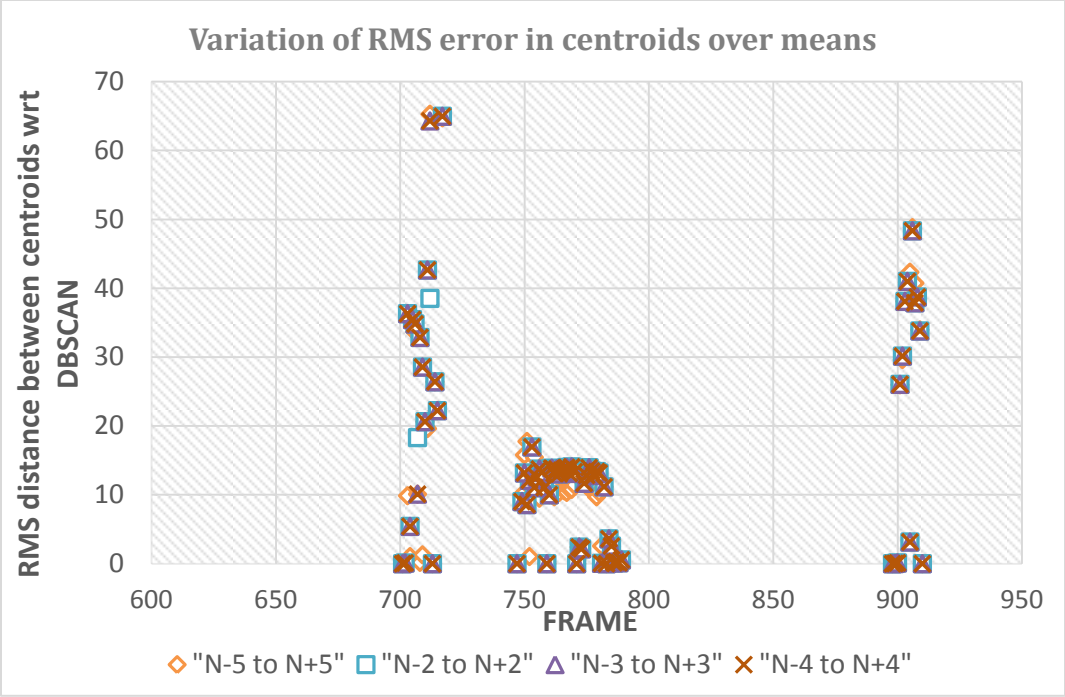


Figure 9: Variation in tracking performance with the number of frames over which K Means is done for multi_car video. Frame parameter is 12.

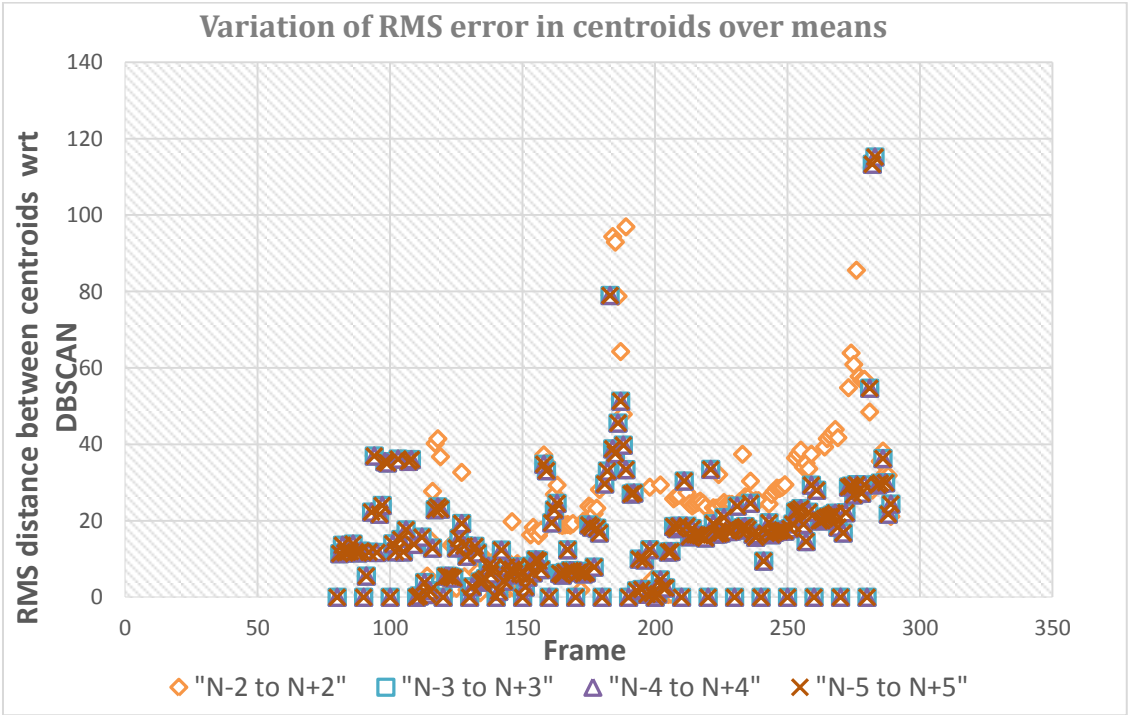


Figure 10: Variation in tracking performance with the number of frames over which K means is done for highway video. Frame parameter is 10.

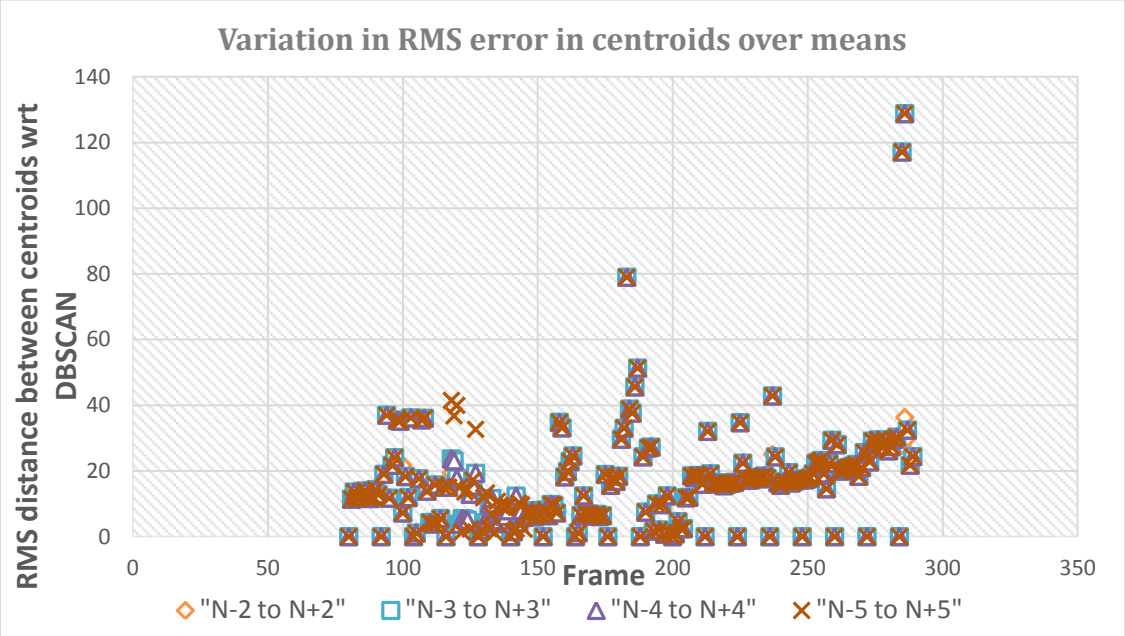


Figure 11: Variation in tracking performance with the number of frames over which K means is done for highway video. Frame parameter is 12

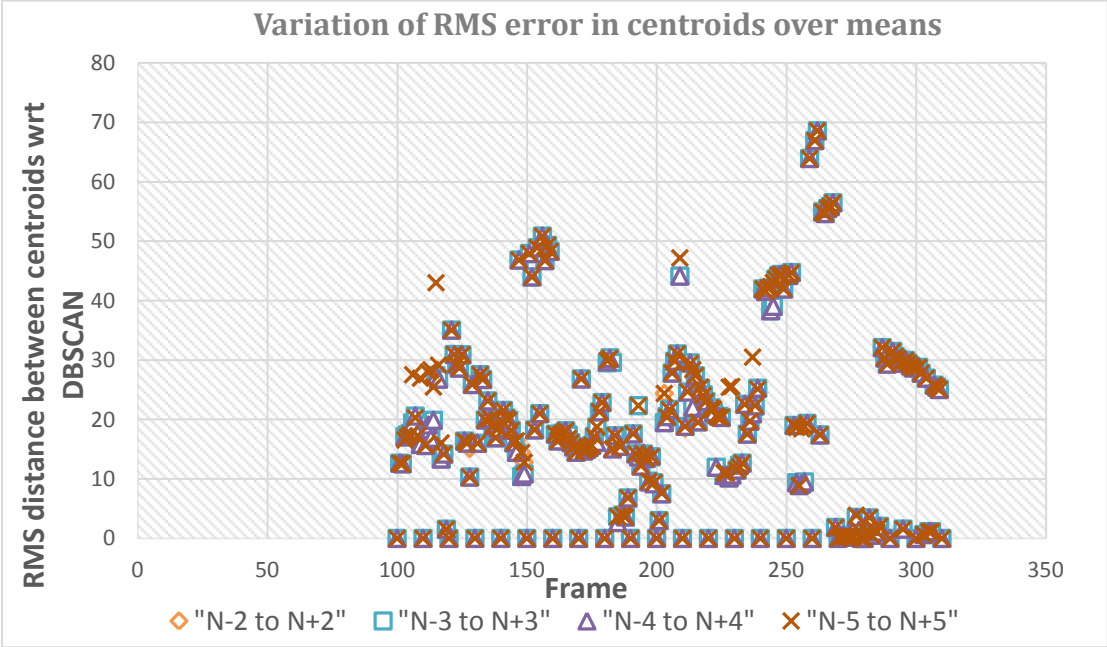


Figure 12: Variation in tracking performance with the number of frames over which K Means is done for Aerial_highway video. Frame parameter is 10.

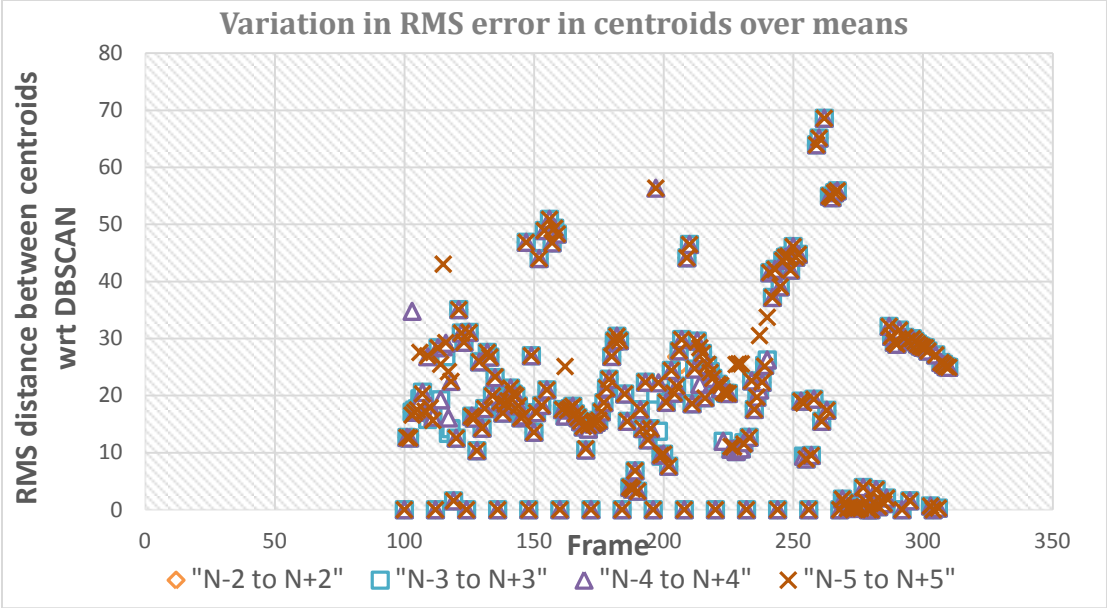


Figure 13: Variation in tracking performance with the number of frames over which K Means is done for Aerial_highway video. Frame parameter is 12.

The best configuration was analyzed for each video based on which had the minimal error. The findings are as below:

Video	Frame Parameter	Number of means
Multi_car	12	N-5 to N+5
Highway	10	N-3 to N+3
Aerial	10	N-4 to N+4

Table 1: Best configuration for each video

4.5 TIMING RESULTS

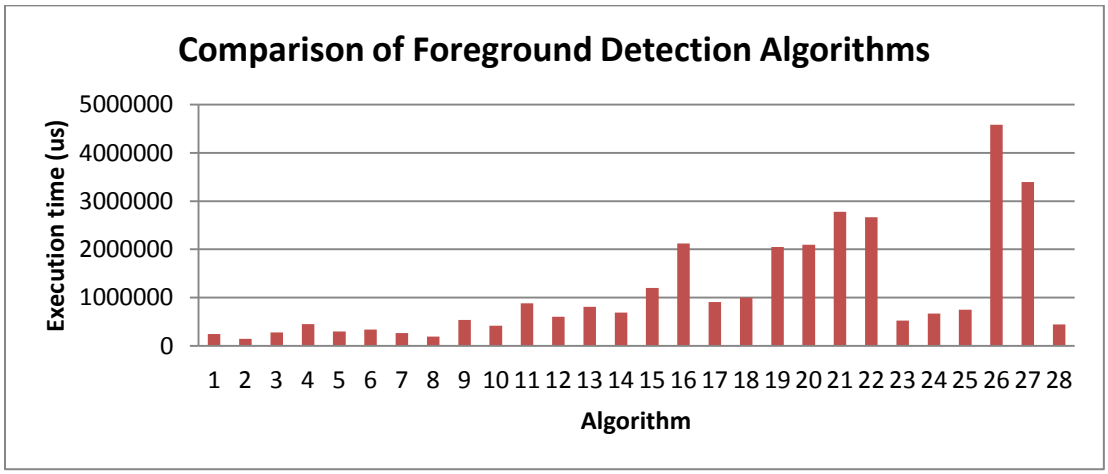
4.5.1 Foreground Detection

a) Execution times of various foreground detection techniques

The implementations of Gaussian mixture model obtained from open source [4] were run on Intel core I7 Processor. The best performing GMM with the minimal number

of computations was picked. The input video was taken from Change detection website [17] was used for this evaluation. A frame of the video is shown in Fig 14.

Figure 14: Input video that was used to find the best foreground detection technique



1	Frame Difference	8	AdaptiveSelectiveBackgroundLearning	15	Temporal Median (Cucchiara&Calderara)	22	FuzzyChoquetIntegral
2	Static Frame Difference	9	GMG (Very poor performance)	16	Eigenbackground (Oliver) (Very Poor performance)	23	LBSimpleGaussian (poor)
3	WeightedMovingMeanBGS	10	DPAdaptiveMedianBGS	17	T2FGMM_UM	24	LBFuzzyGaussian (poor)
4	WeightedMovingVarianceBGS	11	DPGrimsonGMMBGS (Very poor performance)	18	T2FGMM_UV	25	LBMixtureOfGaussians (poor)
5	MixtureOfGaussianV1BGS	12	DPZivkovicAGMMBGS	19	T2FMRF_UM (Poor performance)	26	LBAdaptiveSOM (poor)
6	MixtureOfGaussianV2BGS	13	DPMeanBGS	20	T2FMRF_UV	27	MultiLayer BGS Model
7	AdaptiveBackgroundLearning	14	Gaussian Average (Wren)	21	FuzzySugenoIntegral	28	VuMeter

Figure 15: Variation in execution times of foreground detection algorithm on Intel core I7

Figure 15 shows execution times of different foreground detection algorithms on Intel I7 processor. Among the different algorithms, Zivkovic's GMM was the algorithm of choice (Algorithm 12 in Figure 15). The algorithms which had lesser runtime were not robust enough for tracking purposes. Zivkovic's GMM provided the necessary robustness while having a small runtime.

b) Speedup of Zivkovic's GMM algorithm

Three videos namely Multi-car, Aerial highway and Highway were chosen for applying the parallel version of Zivkovic's GMM. Figure 16 depicts a snapshot of the videos with foreground detection applied.

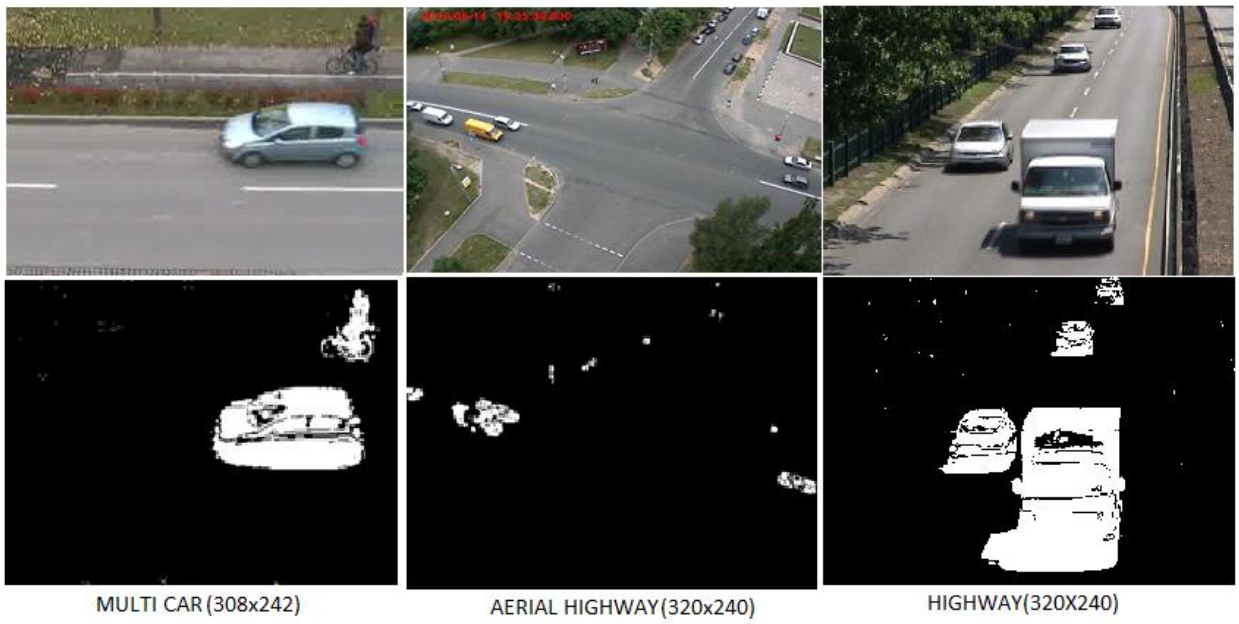


Figure 16: Result of foreground detection algorithm

4.5.2: Clustering

Clustering contributes to a significant portion of the overall runtime. If DBSCAN is used for clustering, then clustering takes ~70% of the overall time.

Performance on Intel Core I7

Video	Execution time for proposed algorithm(us)	Execution time for DBSCAN (us)	Speedup= DBSCAN exe time/ Proposed algorithm exe time
Multi_Car	9460670	39250663	4.15
Aerial_Highway	4904950	12907185	2.63
Highway	10903100	39238625	3.60

Table 2: Performance of DBSCAN on Intel Core I7

Performance on Hardware accelerator:

Video	Execution time for DBSCAN(us)	Execution time for proposed algorithm(us)	Speedup= DBSCAN exe time/ Proposed algorithm exe time
Aerial_Highway	1320000	448457	2.95
Highway	645000	131000	4.92

Table 3: Performance of DBSCAN on hardware accelerator

Table 2 shows the performance of the DBSCAN on Intel Core I7 processor for 210 frames of videos sequences and Table 3 shows the performance of the DBSCAN algorithm on the hardware accelerator for 15 frames. The multi_car video did not run on the hardware accelerator owing to the large memory requirement for the points to be clustered.

The proposed clustering algorithm provides substantial speedups when compared to running DBSCAN for each frame of the video. For Intel Core I7, the average speedup is around 3.9 whereas for the hardware accelerator, the average speedup is around 3.5. The algorithm results in faster runtimes without compromising much on the accuracy provided by DBSCAN.

4.5.3 Kalman Filter

Video	Execution time(us)
Multi car	1400
Highway	994
Aerial_Highway	2248

Table 4: Performance of Kalman filter on hardware accelerator

The Kalman filter runtimes are negligible when compared to the other components of the system. As a result, we directly applied the filter in our system without any modifications. The execution times presented above is the execution time for 15 frames of the three video sequences.

4.5.4. Overall Speedup

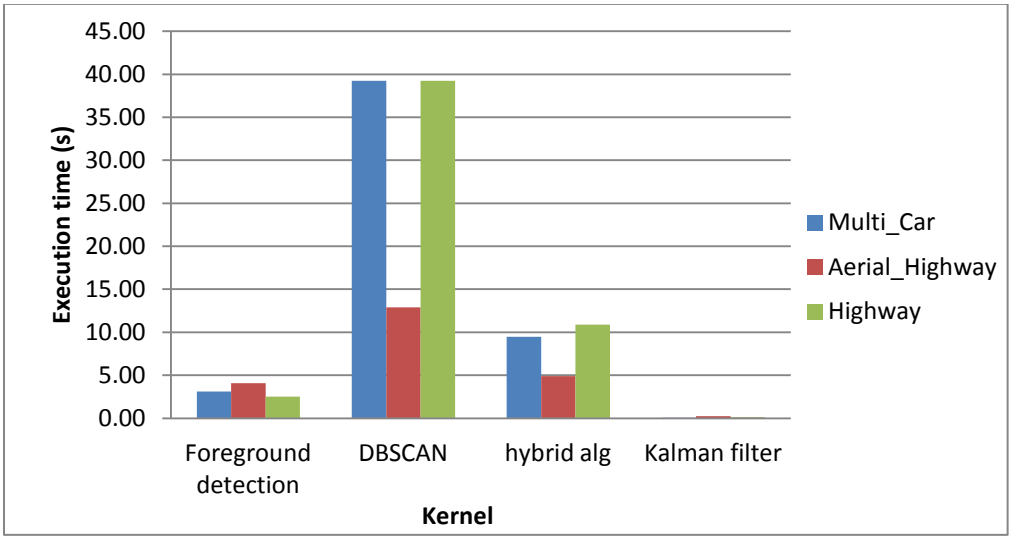


Figure 17: Runtimes of the different kernels on Intel I7.

Figure 17 shows the execution times for the different kernels for 210 frames of the three video sequences on Intel I7 processor. On an average, with the DBSCAN clustering algorithm, the foreground detection was observed to take 12%, DBSCAN 87% and Kalman filter 1% of the total execution time. On the other hand, when the hybrid algorithm was used, foreground detection took 29%, clustering 69% and Kalman filter 2

% of the total execution time. There is a drastic difference in the run times for clustering between Aerial_highway and the rest because the number of data-points to be clustered in the case of Aerial_highway was of the order of 10^2 whereas for the rest, it was of the order of 10^5 and hence more time to cluster.

Video	Exe time (us)		Speedup
	Proposed algorithm	Conventional code using DBSCAN	
Multi car	42417234	12627241	3.4
Highway	17225297	9223062	1.9
Aerial_Highway	41910307	13574782	3.1

Table 5: Overall performance of object tracking system on Intel Core I7

CHAPTER 5

CONCLUSION AND FUTURE WORK

This thesis report described our work on implementing a low cost object tracking algorithm and optimizing it further for a parallel hardware accelerator. The video sequences that were considered were videos captured using stationary cameras. Also, the processing done was offline, i.e processing was done on stored frames from the video. This was because the hardware accelerator simulation setup did not support processing directly on the videos.

We subdivided the task of object tracking into foreground detection, clustering and tracking using 1D Kalman filtering. The choice of our algorithms was made on the basis of computational efficiency without compromising on the accuracy.

For foreground detection, we analyzed a set of algorithms that were a part of the Open Source Software developed by Andrews Sobral [2] by executing them on Intel I7 and checking the computation time as well as visually inspecting the accuracy of the algorithms. We found that the Gaussian Mixture model developed by Zivkovic [4] gave the best performance and also provided scope for parallelization. We experimented with different parallelization schemes, such as pixel level and loop unrolling. We found that pixel level parallelization gave the best speedup for GMM.

Clustering was found to consume the maximum proportion of the execution time since clustering algorithms are inherently very sequential. Both DBSCAN and K-means were considered. Since DBSCAN is computationally more expensive than K-means, we chose a mixed approach wherein DBSCAN was performed on one frame, followed by K-means on subsequent 9 frames. The number of centroids which was input to K-means

was based on the number of centroids returned by DBSCAN. We compared the performance of our hybrid algorithm to DBSCAN per frame and obtained a maximum speedup of 4.92.

For the final task of estimating the motion parameters of the moving objects, we used a constant acceleration model 1D Kalman filter. We did not find a need to parallelize this task as it contributes to a very small fraction of the overall runtime. The parameters that were estimated were position and velocity of the objects.

Also as part of this thesis, we compared our results with the object tracking technique developed by University of Michigan researchers. Their tracking technique works with videos having moving background as well, whereas our technique works only with stationary background.

For future research, we plan to investigate the hybrid clustering algorithm further. There are parallel versions of DBSCAN implemented on GPUs such as the parallel DBSCAN based on Disjoint-Set Data structure [8]. We can take inspiration from that to implement the parallel version on the hardware accelerator.

REFERENCES

[1] H. Zhou, Y. Yuan, C. Shi, "Object tracking using SIFT features and mean shift," Computer Vision and Image Understanding, 113 (3), pp.345‐352, 2009.

[2] A. Sobral and A. Vacavant, "A comprehensive review of background subtraction algorithms evaluated with synthetic and real videos," Computer Vision and Image Understanding, 122, pp.4–21, 2014.

[3] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," ACM Computing Surveys (CSUR), 38(4), pp.1 -45, 2006.

[4] Z. Zivkovic, "Improved adaptive gaussian mixture model for background subtraction," *IEEE International Conference on Pattern Recognition (ICPR),* vol. 2, pp. 28–31, August 2004.

[5] Z.Zivkovic and F.van der Heijden, "Recursive Unsupervised Learning of Finite Mixture Models", *IEEE Trans. on Pattern Analysis and Machine Intelligence,* vol.26, no.5, pp.651-656, May 2004.

[6] H.Bäcklund, A.Hedblom and N.Neijman, "A density - based spatial clustering of application with noise," Data Mining TNM033 LinköpingsUniversitet – ITN, 2011.

[7] T.N. Tran, K.Drab, M.Daszykowski, "Revised DBSCAN algorithm to cluster data with dense adjacent clusters", Chemometrics and Intelligent Laboratory Systems, 120:9296. DOI: 10.1016/j.chemolab.2012.11.006.

[8] M.M.A.Patwary, D.Palsetia, A.Agrawal, W.Liao, F.Manne, A.Choudhary, "A New Scalable Parallel DBSCAN Algorithm using the Disjoint-Set Data Structure", International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2012.

[9] M. Ester, H.P. Kriegel, J. Sander, X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise", Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, Portland, Oregon, pp. 226–231, 1996.

[10] C. Lallier, E. Renaud, L. Robinault, L. Tougne, "A testing framework for background subtraction algorithms comparison in intrusion detection context", 8th IEEE International Conference on Advanced Video and Signalbased Surveillance (AVSS), , pp.314-319, 2011.

[11] M. Sivabalakrishnan, D. Manjula, "Adaptive background subtraction in dynamic environments using fuzzy logic", International Journal of Video & Image processing, IJVIPNS-IJENS, vol:10 No: 01, pp. 18-21, 2010

[12] Wikipedia: https://en.wikipedia.org/wiki/K-means_clustering

[13] P.S. Maybeck, "Stochastic models, estimation, and control", Chapter 1, Introduction, Academic Press, Inc, vol. 1, pp. 1-16, 1979.

[14] G.Welch, G.Bishop, "An Introduction to the Kalman Filter", UNC-Chapel Hill 19(5): 41, 2006.

[15] X.Mei, H.Ling, "Robust visual tracking and vehicle classification via sparse representation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *33*(11), pp.2259–2272, 2011.

[16] B.Babenko, M.-HYang, S.Belongie, "Robust object tracking with online multiple instance learning", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33, pp.1619–1632, 2011.

[17] N. Goyette, P.M. Jodoin, F. Porikli, J. Konrad, and P. Ishwar, "A new change detection benchmark dataset", Proc. IEEE Workshop on Change Detection (CDW-2012) at IEEE Conference on Computer Vision and Pattern Recognition -2012, pp.16-21, June 2012.

[18] J. K Aggarwal, and Q.Cai, "Human motion analysis: A review", *Computer Vision Image Understand. Vol.73, No.*3, pp.428–440, 1999.

[19] D. M. Gavrila, "The visual analysis of human movement: A survey", *Computer Vision Image Understand.Vol.73,* No.1, pp.82–98, 1999.

[20] T.Moeslund, and E.Granum , "A survey of computer vision-based human motion capture", *Comput.Vision Image Understand. 81(* 3), pp.231–268, 2001.

[21] R.O. Duda, P.E. Hart, David G. Stork, "Pattern classification", 2nd edition, Wiley.

[22] J. B MacQueen, "Some Methods for classification and Analysis of Multivariate Observations", Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability **1**. University of California Press. pp. 281–297, 1967.

[23] A. Sobral, L. Oliveira, L. Schnitman, F. Souza, "Highway traffic congestion classification using holistic properties", International Conference on Signal Processing, Pattern Recognition and Applications (SPPRA'2013), DOI: 10.2316/P.2013.798-105, 2013.

[24] J.Shi and C.Tomasi, "Good Features to Track", *IEEE Conference on Computer Vision and Pattern Recognition*, pp.593–600, 1994.

[25] B.D. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision", *Proceedings of the 7th International Joint Conference on Artificial intelligence* pp.674-679,1981.

[26] C.Tomasi and T.Kanade, "Detection and Tracking of Point Features", Carnegie Mellon University Technical Report CMU-CS, pp.91-132, April 1991.

[27] J.Hopcroft, R.Tarjan, "Algorithm 447: efficient algorithms for graph manipulation", Communications of the ACM **16** (6), pp.372–378, 1973.

[28] G. Nebehay, R. Pflugfelder, "Consensus-based Matching and Tracking of Keypoints for Object Tracking", IEEE Winter Conference on Applications of Computer Vision (WACV), pp. 862-869, 2014.

[29] S. Blackman, "Multiple hypothesis tracking for multiple target tracking", IEEE Aerosp. Electron. Syst. Mag., vol. 19, no. 1, pp.5 -18, 2004.

[30] J.Rittscher, J.Kato, S.Joga, and A. Blake, "A probabilistic background model for tracking", European Conference on Computer Vision (ECCV). Vol. 2, pp.336–350, 2000.

[31] B.Resko, P.T Szemes, P.Korondi, P.Baranyi, H.Hashimoto, "Artificial neural network based object tracking," in Annual Conference Society Of Instrument And Control Engineers Of Japan, vol.2, pp.1398-1403, Aug 2004.

[32] A.Babaeian, S.Rastegar, M.Bandarabadi, M.Rezaei, "Mean shift-based object tracking with multiple features," 41st Southeastern Symposium on System Theory, pp.68-72, March 2009.

[33] T. Lindeberg, "Scale Invariant Feature Transform", Scholarpedia, 7(5), 2014.

[34] N. Binkert, B.Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M D. Hill, and D. A. Wood, "The gem5 simulator", ACM SIGARCH Computer Architecture News, May 2011,

[35] J. A.Hartigan, M. A.Wong, "Algorithm AS 136: A K-Means Clustering Algorithm", Journal of the Royal Statistical Society, Series C (applied Statistics), 28(1), pp.100‑108, 1979.

[36] M.A. Fischler , R.C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography", Communications of the ACM, 24(6), pp.381‑395, 1981.

[37] X. Jin and J. Han, "K-Medoids Clustering", Encyclopedia of Machine Learning, Springer US, pp. 564-565, 2010.

[38] T. Zhang, R. Ramakrishnan & M. Livny, "BIRCH: an efficient data clustering method for very large databases", ACM SIGMOD Record ,Vol. 25, No. 2, pp. 103-114, June 1996.

[39] W.Wang, J. Yang, & R. Muntz, "STING: A statistical information grid approach to spatial data mining", *Very Large Data Bases Conference*, Vol. 97, pp. 186-195, August 1997.

[40] G. Sheikholeslami, S. Chatterjee, & A. Zhang, "WaveCluster: a wavelet-based clustering approach for spatial data in very large databases", *The Very Large Data Bases Journal*, *8*(3-4), pp. 289-304, 2000.

[41] http://campar.in.tum.de/Chair/KalmanFilter